Affective Computing - Programming Assignment 1

Objective

The objective of the exercise is to build the facial expression recognition system. The system includes face preprocessing, feature extraction and classification. In the exercise, you will learn how to preprocess the facial expression image, extract the feature from an image or a video, and classify the video into one category.

Specifically, the region of interest (i.e., facial image) is extracted using face tracking, face registration and face crop functions. Basic spatiotemporal features (i.e., LBP-TOP features) are extracted using LBP-TOP. To produce emotion recognition case, Support Vector Machine (SVM) classifiers are trained. 50 videos from 5 participants are used to train the emotion recognition, use spatiotemporal features. The rest of the data (50 videos) is used to evaluate the performances of the trained recognition systems.

Database

The original facial expression data is a sub-set of eNTERFACE (acted facial expression), from ten actors acting **happy** and **sadness** behaviors. The used dataset in the exercise includes 100 facial expression samples.

Task 1. Face preprocessing

In this part, you are supposed to process a face image. There are three subtasks you need to do:

- **Task 1.1.** Detect face and facial landmarks using the `DLib` (http://dlib.net/) library.
- **Task 1.2.** You are asked to perform a face registration task using a set of fixed landmarks from a standard model, and extract face from the registered image.
- **Task 1.3.** Visualize your result using subplots.

Task 1.1. Extract facial landmarks

Steps:

1. Load the example image *exampleImg.jpg*, using the `skimage.io.imread()` (http://scikit-image.org/docs/dev/api/skimage.io.html#skimage.io.imread) function for example.
2. Initialize a face detector and a face landmarks detector. We have provide the code of this part, please learn to use them.
3. Detect face
4. Detect the face landmarks
5. Transfrom the detected result to a 2-D array, we provide the function *shape2points()* below to perform this.

Besides, here is an example for facial landmarks extraction: http://dlib.net/face_landmark_detection.py.html (http://dlib.net/face_landmark_detection.py.html)

In [1]:
```
# Loading required libraries

# We provide the shape to points function.

# 1. Load exampleImg.jpg, using skimage.io.imread()

# 2. Initializing face detector and shape predictor


# 3. Detecting face, return rectangles, each rectangle corresponds to one face.
# You need to fill the missing argument of this function

# Extracting the shape of the face in the first rectangle (using the first element of the rectangles variable

# Extract facial landmarks from shape by calling the shape2points() function.

```

Task 1.2. Face normalization

Steps:

1. Load the landmark position of a standard face model. We provide these positions in a csv file, and also the code block to read these positions.
2. Calculate the transformation between your detected landmarks position and the standard face model landmark positions using the `skimage.transform.PolynomialTransform()` (http://scikit-image.org/docs/dev/api/skimage.transform.html#skimage.transform.PolynomialTransform) class and its *estimate()* methods.
   A. Instantiate a PolynomialTransform object by calling transform.PolynomialTransform()
   B. Call its estimate() method to calculate the transformation between the two sets of points. The manual of this method can be found in the same page which introduced of this class.
3. Transform the example image using the calculated transformation to register (map) the example image into a space of the standard face model. You can use the `skimage.transform.warp()` (http://scikit-image.org/docs/dev/api/skimage.transform.html#skimage.transform.warp) function to perform this.
4. Crop the face from the registered face using the standard face model landmarks. The cropping function is provided in the *exercise1Lib.py*, you can directly use it after importing.
5. Also extract the face from the example image using your detected landmarks.

We offer the code to load a standard face model, which contains the mean position of landmarkds estimated from the training data.

In [2]:
```
# 1. Load the landmark position of the standard face model

```

In [3]:
```
# 2. Calculating the transorfmation between the two set of points
# 2.1 Instantiating a PolynomialTransform() transform function

# 2.2 Calculating the transformation by calling the estimate() method.
#     You do not need to reteurn any value after calling this methods,
#     because the transformation parameter is store in the object you instantiated after calling this methods.

# 3. Warping your example image using the transform.warp() function

# 4. Crop the face from registered image using the provided cropFace function.

# 5. Croping the face from the example image using detected landmarks


```

```
D:\ProgramData\Anaconda2\lib\site-packages\skimage\transform\_warps.py:84: UserWarning: The default mode, 'constant', will be changed to 'reflect' in skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
```

Task 1.3. Display result

Here you are asked to draw a figure with 3 x 2 subplots using `matplotlib.pyplot.subplots()` (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplots.html). Please read the manual of it and also the `matplotlib.pyplot` (https://matplotlib.org/api/pyplot_api.html). Below explains content should be presented in each subplots:

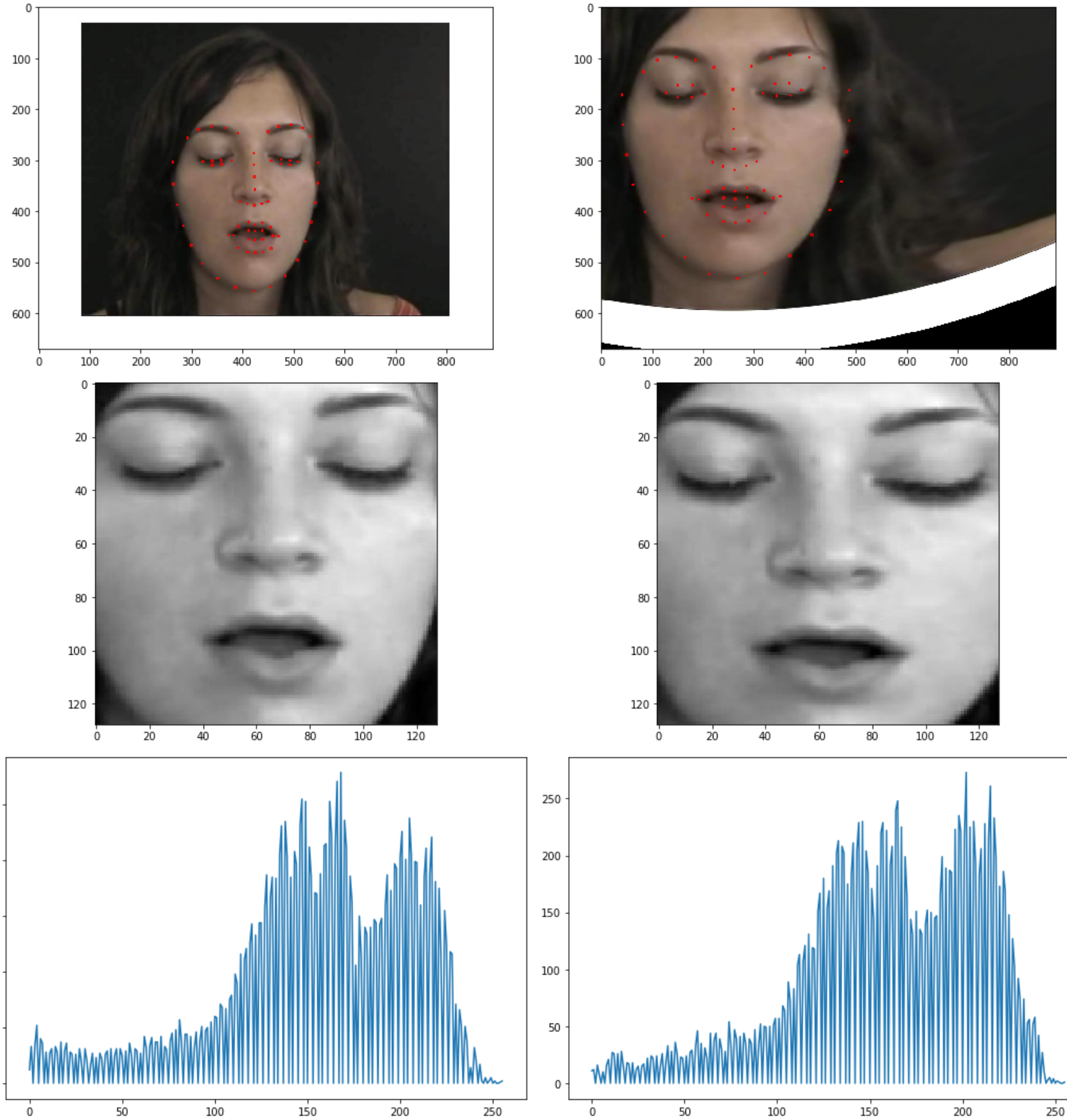
- subplot [0,0]: the original example image and detected landmarks.
- subplot [1,0]: the face cropped from the example image.
- subplot [2,0]: the histogram of the face cropped from the example.

We provide the code for above three subplots, please refer them as examples. Then you need to implemente:

- subplot [0,1]: the registered face image.
- subplot [1,1]: the face cropped from the registered face image.
- subplot [2,1]: the histogram of the face cropped from the registered face image.

In [4]:
```
# Constructing figure with 2x3 subplots

# subplot [0,0]: show the original example image


# Placing detected landmarks on subplot [0,0], we provide an exmaple to do this.


# subplot [1,0]: show the face cropped from the example image.

# subplot [2,0]: show the histogram of the face cropped from the example image.



# subplot [0,1]: show the registered image

# place the model landmarks on the registered image


# subplot [1,1]: show the face cropped from the registered image

# subplot [2,1]: show the histogram of the face cropped from the registered image.

```

```
D:\ProgramData\Anaconda2\lib\site-packages\skimage\util\dtype.py:122: UserWarning: Possible precision loss when converting fro
m float64 to uint8
  .format(dtypeobj_in, dtypeobj_out))
```



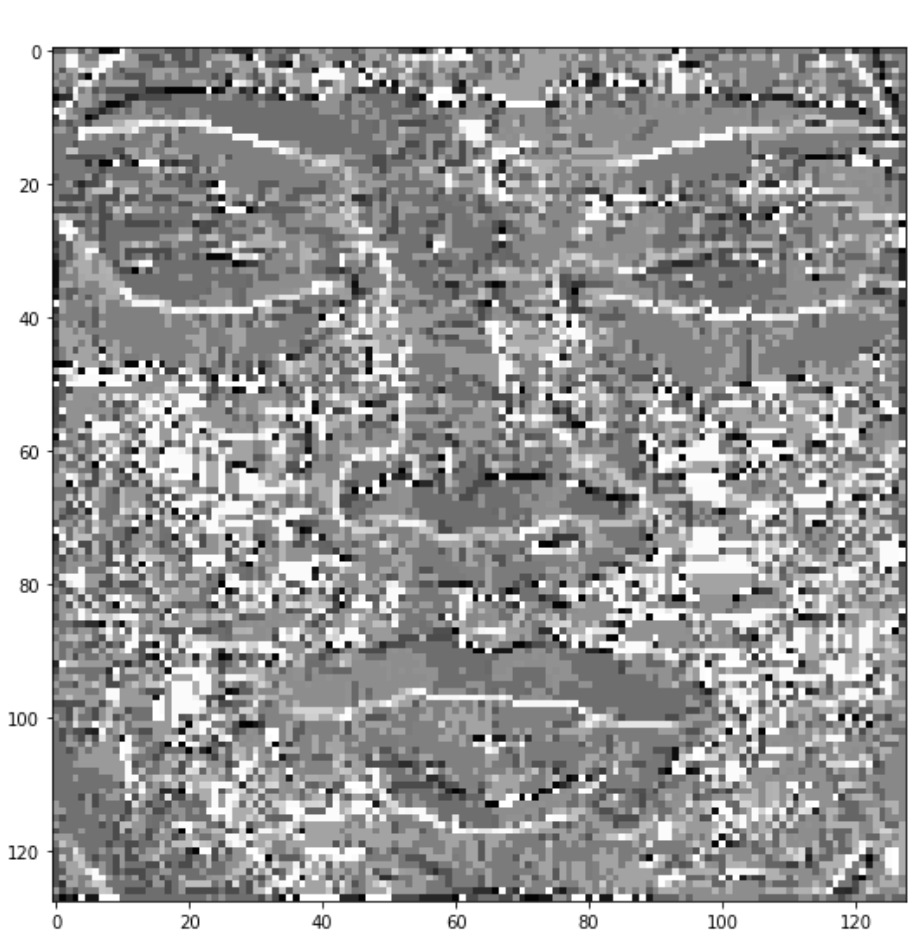
Task 2. Feature extraction

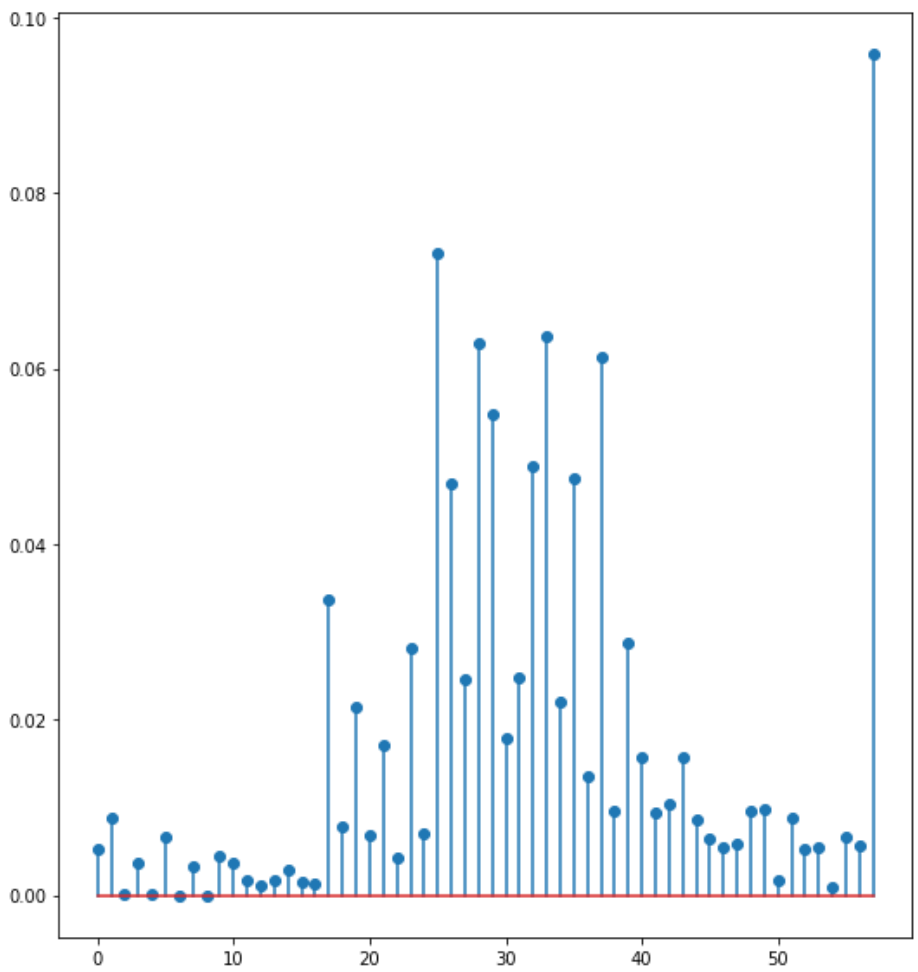
Here you are asked to extract LBP feature. The algorithm is named as Local Binary Pattern. So far many researcher use it in face recognition, facial expression recognition and texture classification. Here you will use the `skimage.feature.local_binary_pattern()` (http://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.feature.local_binary_pattern) function to extract the LBP feature. Steps are explained below.

Steps:

1. Define the LBP parameters. Before doing this, please read carefully of its manual.
   A. P: Number of neighbours, plese set P = 8
   B. R: Radius of circle, please set R = 1.0
   C. LBP methods: please set it as 'nri_uniform'
2. Extract the LBP face by calling the *skimage.feature.local_binary_pattern()* function
3. Calculate the histogram of the LBP face
   A. Caculate the histogram of the LBP face
   B. Normalize the histogram to make the histogram's sum one one (dividing each element by the sum of the histogram).
4. Visualize the result using two subplots.
   A. Draw the LBP face at the left one.
   B. Draw the normalized histogram at the left one, but using *.stem()* function rather *.plot()* for this time.

In [5]:
```
# 1. Define parameter to extract LBP feature in (8, 1) neighborhood:
#    1.1 Please set the number of neighbour P = 8
#    1.2 Please set the radius if circir R = 1.0
#    2.3 Please set the method as 'nri_uniform'

# 2. Extracting the LBP face using local_binary_pattern()

# 3. Calculate the histogram of the LBP face. Sum of vector can be calculated by calling numpy.sum()

# 4. Visualize your result.

```





Task 3. Feature Classification

You will use Support Vector Machine (SVM) for facial expression classification. Please read the `sklearn.svm.SVC()` () and learn to uset it. Mainly you will use its two methods: **fit()** to training the classifier and **predict()** to use the classifer for classification. There are following three subtasks you need to complete:

- Task 3.1. Load data
- Task 3.2. Train classifiers
- Task 3.3. Evaluate classifiers

Task 3.1. Load data

Firstly, you need to read *.mat* files using python. You can use the `scipy.io.loadmat()` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.loadmat.html#scipy.io.loadmat) function to read *.mat* file. In the provided **Task3_data.mat** file, different data are packed by different dictionaries which are list below:

- training_data
- testing_data
- training_class
- testing_class For example if you would like to load the *training_data* from the file, you can use **scipy.io.loadmat('Task3_data.mat')['training_data']**

In [6]:
```
# Loading data using scipy.io.loadmat(), or sio.loadmat
# Alternatively, you can us h5py also, if you would like to

# Load 'training_data'

# Load 'testing_data'

# Load 'training_class'

# Load 'testing_class'

```

Task 3.3. Train SVM classifiers

Use the **sklearn.svm** library to train Support Vector Machine (SVM) classifiers. The 'training_data' and 'testing_data' matrices contain the calculated LBP-TOP features for the training and testing sets, respectively. The block size for LBP-TOP used for training and testing data are 2x2x1. The 'training_class' group vector contains the class of samples: 1 = happy, 2 = sadness, corresponding to the rows of the training data matrices.

Steps:

1. Construct an SVM classifier using the linear kernel. Please read `sklearn.svm.SVC()` (http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC).
2. Use the `fit()` (http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.fit) method and the *training_data* and *training_class* to train your classifier.

In [7]:
```
# 1. Initializing a SVM classifier, using linear kernel

# 2. using the classifier to fit your training data

```

Out[7]:
```
SVC(C=1.0, cache_size=5000, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

Task 3.3. Evaluate your classifiers

Steps:

1. Use your trainined classifer to classify the *training_data* and *testing_data*, using the `predict()` (http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.predict) method.
2. Calculate the classification accuricies when classifying the *training_data* and *testing_data*, respectively. The correct class labels corresponding with the rows of the training and testing data matrices are in the variables *training_class* and *testing_class*, respectively.
3. Calculate the confusion matrices when evaluating either dataset. Using `sklearn.metrics.confusion_matrix()` (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html).

In [8]:
```
# 1. Predicting you training data and testing data.

# 2. Calculating the accuracies of your prediction on training data and testing data, respectively.
#    2.1 calculate the accuracy when classifying the training data

#    2.2 calculate the accuracy when classifying the test data


# 3. Draw your confusion matrix
# 3. using sklearn.metrics.confusion_matrix
#    3.1 Calculate the confusion matrix when classifying the training data


#    3.2 Calculate the confusion matrix when classifying the testing data

```

```
0.88
0.72
[[20  5]
 [ 1 24]]
[[25  0]
 [14 11]]
```